**IT Infrastructure**

**Dr. Usman**

**Name:** Muhammad Saad **Sap ID:** 22183 **Program:** MSCS

***Final Assignment***

**Question#1:** **What is the test automation framework?**

A **Test Automation Framework** is a set of guidelines like coding standards, test-data handling, object repository treatment etc. Which when followed during automation scripting produces beneficial outcomes like increased code re-usage, higher portability, reduced script maintenance cost etc. These are just guidelines and not rules; they are not mandatory and you can still script without following the guidelines. But you will miss out on the advantages of having a Framework [1].

**Why do you need it?**

Let’s consider an example to understand why you need a Framework.

I am sure you have attended a seminar/lecture/conference where the participants were asked to observe the following guidelines –

* Participants should occupy their seat 5 minutes before the start of a lecture
* Bring along a notebook and pen for note taking.
* Read the abstract so you have an idea of what the presentation will be about.
* Mobile Phones should be set on silent
* Use the exit gates at opposite end to the speaker should you require to leave in the middle of the lecture.
* Questions will be taken at the end of the session

Do you think you can conduct a seminar WITHOUT observing these guidelines????

The answer is a big YES! Certainly, you can conduct a seminar / lecture / conference / demonstration without the above guidelines (in fact some of us will not follow them even though there are laid …

But if the guidelines are followed it will result in a beneficial outcome like reduced audience distraction during lecture and increased participant retention and understanding of the subject matter.

Based on the above, a **Framework can be defined as a set of guidelines which when followed produces beneficial results.**

**How does it work?**

A testing framework is a set of guidelines or rules used for creating and designing test cases. A framework is comprised of a combination of practices and tools that are designed to help QA professionals test more efficiently. These guidelines could include coding standards, test-data handling methods, object repositories, processes for storing test results, or information on how to access external resources [2].

**What is selenium?**

**Selenium** is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts. Testing done using the Selenium testing tool is usually referred to as Selenium Testing [3].

**Why do you need it?**

Imagine that a manual tester has this scenario: Checking whether the web app’s signup page (www.example.com/signup) validates input strings and registers a user successfully in latest versions of Chrome and Firefox, on Windows 7.

Assume that the signup page has these input fields—username, email address, and password. The tester will get a Windows 7 desktop and follow these steps, consecutively, on latest versions of Chrome and Firefox:

1. Enter the URL in the address bar (www.example.com/signup)
2. Enter an invalid string in each input field (email, username, and password)
3. Check whether the input strings were validated against corresponding regexes and any pre-existing values in the database
4. Enter ‘valid’ strings in each input field; click Sign Up
5. Check whether “Welcome, ‘{‘username’}’“page showed up
6. Check whether the system database created a new userID for ‘{‘username’}’
7. Mark the test ‘passed’ if it did, ‘failed’ if the signup feature broke anywhere during the test.

That’s a very basic system test. In the real world, testers are more likely to be checking all user workflows on www.example.com for breakage, on as many OS-browser combinations as needed to meet the benchmarked compatibility standards.

Depending on the number of manual testers (and thoroughness of test cases), it may take anywhere between hours to weeks to be sure that the web app is fully functional.

Modern developers and product teams don’t have that kind of time to allot for testing, but they can’t set aside exhaustive testing in a hurry to release either. This is why they super-charge their testing with automation, powered by Selenium [4].

**How does it work?**

The WebDriver protocol has a local end (‘client’) which sends the commands (test script) to a browser-specific driver. The driver executes these commands on its browser-instance. So, if the test script calls for execution on Chrome and Firefox, the Chrome Driver will execute the test on Chrome; the Gecko Driver will do the same on Firefox.

**Question#2: The most common tools that are used for configuration management are packer and ansible. You need to concisely compare both of them**.

**Configuration Management (CM)** can be defined as the process of systematically handling changes to a system in a way that it maintains consistency among your assets in that system over time. The purpose of it is to identify and track individual configurations items, documenting their functional capabilities as well as their interdependencies and figure out how best to exploit them [6].

**What is Ansible?** Radically simple configuration-management, application deployment, task-execution, and multi-node orchestration engine. Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates. Ansible’s goals are foremost those of simplicity and maximum ease of use.

**What is Packer?** Create identical machine images for multiple platforms from a single source configuration. Packer automates the creation of any type of machine image. It embraces modern configuration management by encouraging you to use automated scripts to install and configure the software within your Packer-made images.

Some of the features offered by Ansible are:

* Ansible's natural automation language allows sysadmins, developers, and IT managers to complete automation projects in hours, not weeks.
* Ansible uses SSH by default instead of requiring agents everywhere. Avoid extra open ports, improve security, eliminate "managing the management", and reclaim CPU cycles.
* Ansible automates app deployment, configuration management, workflow orchestration, and even cloud provisioning all from one system.

On the other hand, Packer provides the following key features:

* Super fast infrastructure deployment. Packer images allow you to launch completely provisioned and configured machines in seconds, rather than several minutes or hours.
* Multi-provider portability. Because Packer creates identical images for multiple platforms, you can run production in AWS, staging/QA in a private cloud like OpenStack, and development in desktop virtualization solutions such as VMware or VirtualBox.
* Improved stability. Packer installs and configures all the software for a machine at the time the image is built. If there are bugs in these scripts, they'll be caught early, rather than several minutes after a machine is launched.
* **"Agentless"** is the top reason why over **251** developers like Ansible, while over **24** developers mention **"Cross platform builds"** as the leading cause for choosing Packer.
* Ansible and Packer are both open-source tools. It seems that Ansible with **38.2K** GitHub stars and **16K** forks on GitHub has more adoption than Packer with **9.1K** GitHub stars and **2.47K** GitHub forks.
* **DigitalOcean**, **9GAG**, and **Rainist** are some of the popular companies that use Ansible, whereas Packer is used by **Instacart**, **Oscar Health**, and **Razorpay**. Ansible has a broader approval, being mentioned in **960** company stacks & **587** developers stacks; compared to Packer, which is listed in **115** company stacks and **21** developer stacks [5].

**Picking configuration management tools**

Picking the best set of tools for your team and organization is integral to the success of adopting a configuration management system. There are many tools on the market each with a different set of features and different complexity levels. When choosing a tool, these are some of the things to take into consideration;

* **Your needs**  
  It is important to have a comprehensive understanding of the problem you want to solve with the tools that way your search is for the problem-solving tool not just any CM tool.
* **Inclusivity**  
  The tools should be inclusive and enable full collaboration between your teams as well as enable them to use their own tools should the need arise.
* **Infrastructure complexity**  
  Depending on the size of the project or organization, the level of infrastructure complexity will vary, however it is important to take into consideration aspects like scalability and security, which may not be enforced by the tool.
* **Learning Curve**  
  The speed at which your team takes to get comfortable using the tools will depend on the infrastructure requirements as well the tools custom syntax. How long it takes to see a return on investment is ultimately dependent on how long the team takes to get comfortable with the tools.

## Cost You can only acquire what you can afford. Cost in terms of money to pay for subscriptions as well as time spent to train your team or the cost of hiring people that already possess the necessary knowledge and skills. **Packer and Ansible together**

We are going to use Packer as a provisioner and Ansible for configuration management to deploy a Python/Flask API to AWS to create an[Amazon Machine Image](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html) (AMI) as mentioned earlier.

Packer and other provisioning software make it easier to ensure that you are running the same software from development to production without worrying about changes sneaking in.

To get started install [Packer](https://www.packer.io/) and [Ansible](https://www.ansible.com/) on your computer. I used [Homebrew](https://brew.sh/) to install it on a Macbook, depending on the OS you’re working with, the way to do this might vary.  
Run these commands to install:  
brew install ansible  
brew install packer

Next, we will create the packer template.

Packer template — This is a json file that defines one or more builds by configuring the various components of Packer. Packer reads the template and uses those settings to create multiple machine images in parallel. In our case we will only be configuring one image so we will not get to see the creation in paralle implemented. Perhaps I’ll come back to this article and include it when I’m not caught for time.

The template

{  
"variables": {  
 "access\_key": "{{env `aws\_access\_key`}}",  
 "secret\_key": "{{env `aws\_secret\_key`}}",  
 "ami\_id": "{{env `ami\_id`}}",  
 "region": "{{env `region`}}"  
 },  
"builders": [  
 {  
 "type": "amazon-ebs",  
 "region": "{{user `region`}}"  
 "access\_key": "{{user `access\_key`}}",  
 "secret\_key": "{{user `secret\_key`}}",  
 "source\_ami": "{{user `ami\_id`}}",   
 "instance\_type": "t2.micro",   
 "ssh\_username": "ubuntu",   
 "ami\_name": "cp3-17-{{isotime | clean\_ami\_name}}",   
 "ami\_description": "CP3 ansible packer AMI with Ubuntu 16.04 instance",  
 "tags": {  
 "role": "python-api-17-12-17"   
 },   
 "run\_tags": {  
 "role": "buildSystem"   
 }   
 }  
 ],  
"provisioners": [  
 {  
 "type": "ansible",  
 "playbook\_file": "./pacsible-playbook.yml"  
 }  
 ]  
}

variables: This is where you define custom user variables, in this case the ami\_id, the region and Amazon Web Services (AWS) access key and secret key. The AWS keys can be found on AWS under IAM users Security Credentials. Add them to your environment variables and they will be picked up by packer. The ami\_id is the id of the source\_ami and the region is the region in which the instance should be created. These will be passed to the template when running the build using the var parameter.  
NOTE: Do not explicitly add your AWS credentials in the packer file if you plan to share it anywhere. These credentials can be used to spin up images on your account as you will soon see so sharing them publicly is a recipe for disaster.

builders: This controls what base system Packer builds on and follows the settings in this section to create the machine image.  
In our builder we have:

* type which tells packer to create Amazon AMIs backed by EBS volumes for use in [EC2](https://aws.amazon.com/ec2/).
* region The name of the region to launch the EC2 instance to create the AMI.
* access key and secret key previously mentioned in the variables section which are used to communicate with AWS.
* source\_ami is a pre-existing image which is used as the base for the image we are going to create.
* instance\_type is the EC2 instance type to use while building the AMI.
* ami\_name is a name to identify the Image. We use the isotime variable pick up the time at the point of creating the image and append it to the name to ensure that the name is unique and the clean\_ami\_name variable to remove any characters from the time that are not allowed in the name.
* tags applied to the AMI
* run\_tags applied to the instances launched to create the image for easy identification.

[**provisioners**](https://www.packer.io/docs/provisioners/index.html)**:** In this section you can have an array of all the components that install and configure software within a running machine before it is turned into a static image. They ensure that the image has all the necessary software. In our provisioners section we only have one.  
Each provisioners definition is in form of a json object with keys used to configure the provisioner. The type key is expected and specifies the name of the provisioner to use, in our case this is Ansible. The playbook\_file tells packer which file is to be executed by Ansible.

**Reference:**

1. https://www.guru99.com/test-automation-framework.html

2. https://smartbear.com/learn/automated-testing/test-automation-frameworks/#:~:text=A%20testing%20framework%20is%20a,QA%20professionals%20test%20more%20efficiently.

3. https://www.guru99.com/introduction-to-selenium.html

4. https://www.browserstack.com/selenium#what-do-I-need-to-get-started

5. https://stackshare.io/stackups/ansible-vs-packer

6. https://medium.com/@Thegaijin/configuration-and-change-management-with-packer-and-ansible-f0a16677e28f